# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | August 28, 1998 | Final |

4. TITLE AND SUBTITLE
The Commercialization of a Rapid Prototyping Development Tool for Real-Time Embedded Software Intensive, Process, and Resource Management Systems

5. FUNDING NUMBERS
DAAH04-96-C-0001

6. AUTHORS
Dr. J. Kent Haspert
Mr. Anthony Beauregard

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Illgen Simulation Technologies, Inc.
130 Robin Hill Road, Suite 200
Goleta, CA 93117

8. PERFORMING ORGANIZATION REPORT NUMBER
IST98-R-198

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
Attn: AMXRO-AAA
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 34923.1-MA-ST2

11. SUPPLEMENTARY NOTES

This work was sponsored by the U.S. Army Research Office, Code AMXRO-AAA, under DAAH04-96-C-0001

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; STTR Phase II final report, distribution unlimited.

12B. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

CAPS is a rapid prototyping tool, developed by the Naval Postgraduate School (NPGS) as a graduate research program between 1990 and 1994. The version of CAPS initially provided to ISTI by the NPGS had a number of problems since it was developed and modified by several different graduate students as their thesis project, and thus was subject to constant modification with little documentation, configuration control, or standardization. Under the STTR, ISTI developed a thorough understanding of the CAPS code, was successful in getting it up and running and successfully applied it to solve a "real world" problem. During the course of the Phase I and II STTR program, ISTI was successful in addressing and correcting a number of the problems that resulted in the following significant improvements:

- Revision of the PSDL editor –intermediate language used within CAPS to convert user-input system descriptions into executable code.
- Implementation of library lookup feature – CAPS was intended to access previously developed CAPS routines. ISTI developed this capability and set it up so that non-CAPS routines could also be incorporated into CAPS-generated executable software.
- Modification of time-scale/resolution inherent in CAPS –CAPS version delivered to ISTI supported millisecond time resolution; not suitable for all applications.

14. SUBJECT TERMS
Computer-Aided Prototyping System (CAPS)
Iterative/Adaptive Graphically driven Interfaces
Evolution of Software-Intensive Systems

15. NUMBER OF PAGES 22

16. PRICE CODE

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | None |

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std 239-8
298-102

Copy ____ of ____

# The Commercialization of a Rapid Prototyping Development Tool for Real-Time Embedded Software Intensive, Process, and Resource Management Systems

**IST98R-198**

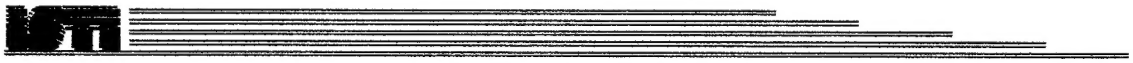
**August 1998**

**Prepared Under Contract**
**DAAH04-96-C-0001**



**Prepared for:**
**Department of the Army**
**Army Research Office**
**P.O. Box 12211**
**Research Triangle Park, NC 27709-2211**

**Prepared by:**
**Illgen Simulation Technologies, Inc.**
**130 Robin Hill Road, Suite 200**
**Goleta, CA 93117**



# Contents

## List of Illustrations



## List of Tables

## Executive Summary

This report constitutes the final report and summarizes the activities performed by Illgen Simulation Technologies, Inc. (ISTI) under a Small Business Technology Transfer (STTR) contract with the US Army Research Office (Contract No. DAAH04-96-C-0001) to develop and commercialize the Computer Aided Prototyping System (CAPS).

CAPS is a rapid prototyping tool, developed by the Naval Postgraduate School (NPGS) as a graduate research program between 1990 and 1994. The version of CAPS initially provided to ISTI by the NPGS had a number of problems since it was developed and modified by several different graduate students as their thesis project, and thus was subject to constant modification with little documentation, configuration control, or standardization. Under the STTR, ISTI developed a thorough understanding of the CAPS code, was successful in getting it up and running, and successfully applied it to solve a "real world" problem. During the course of the Phase I and Phase II STTR program, ISTI was successful in addressing and correcting a number of the problems that resulted in the following significant improvements:

- Revision of the PSDL editor – the intermediate language used within CAPS to convert user-input system descriptions into executable code.
- Implementation of the library lookup feature – CAPS was always intended to have the ability to access previously developed CAPS routines. ISTI developed this capability and set it up in such a way that non-CAPS routines could also be incorporated into CAPS-generated executable software.
- Modification of the time-scale/resolution inherent in CAPS – The CAPS version delivered to ISTI only supported millisecond time resolution, which is not suitable for all applications.
- Enhancement of CAPS' ability to model interdependencies – The originally provided version of CAPS only supported a static dependence of the execution time of one process on the status of some other process. ISTI changed this to a dynamic dependency.
- The elimination of all proprietary (third party) code.

ISTI has documented additional enhancements required to make CAPS a commercially viable tool, and plans to provide CAPS-related services available to the public through its web-based modeling and simulation-consulting network, SimCentral[SM].

# 1.0 BACKGROUND

This section provides a brief history of CAPS, summarizes the contractual arrangement under which Illgen Simulation Technologies, Inc. (ISTI) developed and enhanced CAPS, and describes the key issues that ISTI addressed as a part of the CAPS development effort.

## 1.1 History of CAPS

The Computer-Aided Prototyping System, or CAPS, was originally developed during the early 1990s at the Naval Postgraduate School (NPGS) in Monterey, CA as a graduate level research project. After its initial development, it underwent several improvements and modifications as individual research projects by graduate students. In 1994, ISTI learned that the government had a desire to convert this academic capability into a viable commercial product. ISTI, in conjunction with Monmouth University submitted a Phase I Small-Business Scientific Technology Transfer (STTR) proposal to the Army Research Office (ARO). As a result, a Phase I STTR contract was issued to ISTI in September 1994. A follow-on Phase II development contract was subsequently issued in October 1995. This report summarizes the activities under these efforts, with primary emphasis on the second Phase.

The CAPS approach to rapid prototyping combines the best features of a computer-aided software tool and human-in-the-loop simulation. It provides a mechanism for the real user to work directly with the programmer to refine software performance requirements through an iterative, build-a-little, test-a-little process, which generates ADA code for rapid insertion into real equipment platforms. CAPS is unique in that it has a number of architectural elements that address key functions needed in software development. Table 1-1 describes the requirements for a successful general-purpose software development tool for real-time embedded systems.

The Phase I STTR contract, received in September 1994, established the framework, to include user feedback and documentation, and identified the necessary requirements for a versatile tool that has both military and commercial applications. ISTI received the Phase II portion of this STTR Program October 1995. The Phase II program has dealt with the research and development necessary to yield a well-defined deliverable product suitable for commercialization and export to other potential user groups. This STTR was funded by the Army Research Office (ARO) due to extensive investment on its part at NPGS in CAPS and its desire to achieve a technology transfer from the laboratory into industrial and government applications.

The objective of the Phase II project was to create a highly marketable, rapid prototyping development tool capable of prototyping software intensive systems while supporting the development and evolution of the system throughout numerous iterations. To evolve CAPS into commercial product, the ISTI team established the following technical objectives:

- Create a user-friendly, fully documented, tested, and supported software development tool with an initial focus on rapid prototyping of a software intensive system. The tool would be capable of producing ADA, C, and C++ code from a system specification

## Table 1-1. Prototyping Tool Requirements.

| Requirement | Function |
|---|---|
| ***The CASE tool should implement a software development environment that supports the entire life cycle process, from software design and testing through revision and final product release.*** | ***The tool should assist the software designer/analyst in efficiently and effectively developing a software end product.*** |
| 1. The tool environment should support configuration management and change control functions. | Configuration management and change control functions will provide the user with the methods and tools for:<br>♦ Identifying, storing and assessing software components.<br>♦ Developing a documentation repository.<br>♦ Automating support for change requests.<br>♦ Generating a development history audit trail.<br>♦ Examining traceability between components and product evolution. |
| 2. The tool environment should support project management functions. | Project management functions will provide the user with the methods and tools for:<br>♦ Automatically generating project status reports.<br>♦ Automatically rescheduling project workplans.<br>♦ Automatically briefing programmers. |
| 3. The tool environment should support a common/shared repository for the maintenance of design documentation and software code. | A common/shared repository of design documentation and software code will provide the user with:<br>♦ Version control over both software code and documentation.<br>♦ Global consistency and completeness checking. |
| 4. The tool environment should consider human factors. | Human factors consideration will allow the user to:<br>♦ Access on-line system and team member interaction.<br>♦ Eliminate complex commands.<br>♦ Intuitively assess system operation.<br>♦ Access on-line diagnostics and on-line help. |
| 5. The tool environment should support real-time application functions. | Real-time application functions will provide the user with:<br>♦ Methods and tools for generating real-time system schedules. |
| 6. The tool environment should support automatic code generation from a graphical specification language | Automatic code generation from a graphical specification language will allow the user to:<br>♦ Generate a pictorial view of the software.<br>♦ Automate software generation tasks to produce Ada, C, C++ code<br>♦ Offer predesigned applications |
| 7. The tool environment should support the characterization and reuse of software modules with a populated database. | Software module characterization and reuse functions will:<br>♦ Reduce programming workload.<br>♦ Increase programmer competence. |
| 8. The tool environment should support rapid integration functions for the integration of developed software with other software packages. | Rapid integration functions will provide the user with a wide spectrum of tools for:<br>♦ Automating software integration tasks. |
| 9. The tool environment should support rapid software prototyping and design testing functions. | Prototyping and design testing functions will provide the user with the methods and tools for:<br>♦ Rapid functional prototyping, performance prototyping and behavioral prototyping.<br>♦ Rapid testing of real-time systems. |
| 10. The tool environment should support flexibility and extensibility. | Flexibility and extensibility provides the user the ability to:<br>♦ Use and work with different operating systems and workstation environments. |

written in a prototyping system design language with only minor augmentation on the part of the software developer. This would be true of real-time embedded applications as well.

- Provide a required software support environment assisting in the design, implementation testing, configuration management, and documentation of CAPS. The environment will contain a series of software development commercial-off-the-shelf tools and will employ current procedures and standards used by members of the ISTI team.
- Develop a library of ADA, C and C++ objects that will permit reuse of code in a broad cross-section of applications. These objects would also be employed in a number of applications that will be offered with the product for government, industry, and university use.
- Make provisions for a series of B test sites initiated in Phase I of this effort, for user tests and building additional applications.

The CAPS commercialization plan consisted of four related sections. They included a technical plan, management plan, marketing plan, and product support plan. The first part of the technical plan was to include:

- A series of new version releases creating a progression from the CAPS Version 1.0 software to a fully commercialized version.
- The goal of the second part was to enhance the set of tools and procedures required fully supporting this CAPS evolution.
- The final part of the technical plan consisted of utilizing the Beta test sites to interactively test the system.

Key elements of the management plan included:

- The focus of configuration management, documentation, software development, and product support was to be conducted at ISTI and by the consultants.
- The focus of application development and acceptance testing was to be performed at Monmouth University (MU) in New Jersey and in Monterey, California, where the consultants were located. Integration was to be performed at ISTI.
- Beta site testing was to be conducted in several locations. This included MITRE and CECOM for government applications, AT&T Bell Laboratories and Telos for domestic applications, and MU and Georgia Tech for university use.
- An extensive commitment from ISTI and MU involving their own funding, personnel support, and equipment use. These funds were in excess of the dollars funded by the STTR Phase I effort. This represented a 30% augmentation to the STTR Phase II effort.

CAPS marketing objectives included the following:

- Offer CAPS as a commercial product either directly by ISTI or through a software distributor, e.g., as a product licensed to, say, Silicon Graphics.
- Emphasize non-government application and sales.

Product support was to be offered by:

- On-line help built into the CAPS software.
- A telephone hotline offered at ISTI and MU.
- Release of product with supporting documentation, including a user's manual, training manual, and related specification sheets.

The statement of work for subcontractors proposed that the Man Machine Interfaces (MMI) were to be enhanced by including a more user-friendly graphics editor and graphics viewer, an improved data protection/mistake prevention/trusted system, and a more user-friendly view of timing data. The SOW also proposed that improvement should be made to achieve easy installation/maintenance/use and automating TAE+ script files to achieve a rapid use of the application environment.

The subcontractors' efforts dealt with resolution of a series of problems with the CAPS software identified in Phase I of this project and implementing a limited number of enhancements designed to make the current core capabilities of CAPS more usable by future customers.

## 1.2 Contractual Relationships

CAPS was developed under a multi-phase STTR effort. The Phase I effort was awarded on September 30, 1994, by the US Army Research Office under Contract No. DAAH04-94-C-0053. This effort ran until June 29, 1995. Monmouth University served as a subcontractor and a consultant supported this Phase I effort. The Phase II STTR effort was awarded on October 10, 1995, by the US Army Research Office under Contract No. DAAH04-96-C-0001. This effort originally was scheduled to run until October 15, 1996. However, because of some legal issues associated with the original version of CAPS provided to ISTI, this effort was expanded and extended to June 30, 1998. Monmouth University and several consultants also supported this Phase II effort.

## 1.3 CAPS Development Issues

The version of CAPS provided to ISTI for development and commercialization under the STTR effort presented several significant issues that ISTI had to address. They involved the Graphical User Interface (GUI) development tools used by CAPS, the Prototype Software Development Language (PSDL) synthesizer embedded in the government-furnished version of CAPS, and the relationship with and performance of the government-directed independent consultants. These issues are discussed further below:

- The initial CAPS development at the NPGS used a GUI development tool called TAE+. Unfortunately, the TAE+ tool was no longer in active development and was not commercially supported. This has meant that ISTI could not readily improve the original GUI design. The only option was to totally rebuild the CAPS GUIs, but the STTR contact did not provide adequate resources for this effort. A later section of this report

will discuss why rebuilding the GUIs would have been a worthwhile effort and how ISTI is considering addressing this in the future.

- The PSDL synthesizer in the government-furnished version of CAPS needed to be replaced because it consisted of proprietary software.

# 2.0 DESCRIPTION OF CAPS

This section describes the features incorporated in CAPS, the operation of CAPS, and an assessment of the benefits and current limitations of CAPS.

## 2.1 Features

The CAPS computer-aided prototyping tool automates the initial software design process. Prototyping is an important technique for developing complex, embedded systems that can have strict time constraints. The prototyping technique can also be used to quickly develop computer simulations of systems and processes. The CAPS approach offers a true prototyping language and architectural framework that combines the best features of a computer-aided software tool and human-in-the-loop simulation development. It supports the refinement of requirements via the iterative:

- build-a-little,
- test-a-little

software-development paradigm.

CAPS supports concept confirmation via rapid software creation and testing involving the ultimate user. Also, CAPS can generate Ada code for direct insertion into real equipment platforms. CAPS will soon generate code in other languages, such as C++. CAPS can accomplish this via its

- graphical user interface,
- reusable software database system,
- flexible execution-support process.

Figure 2-1 illustrates the CAPS graphical user interface for entering system descriptions. This figure shows how the various components of a system and the interconnections between these components can be entered into CAPS. This graphical-input process forces structured representations of systems.

CAPS' power lies in its ability to automatically

- generate executable real-time software,
- analyze the design of the software's scheduling routines,
- generate real-time processor scheduling information,
- provide output code once the scheduling information exists.

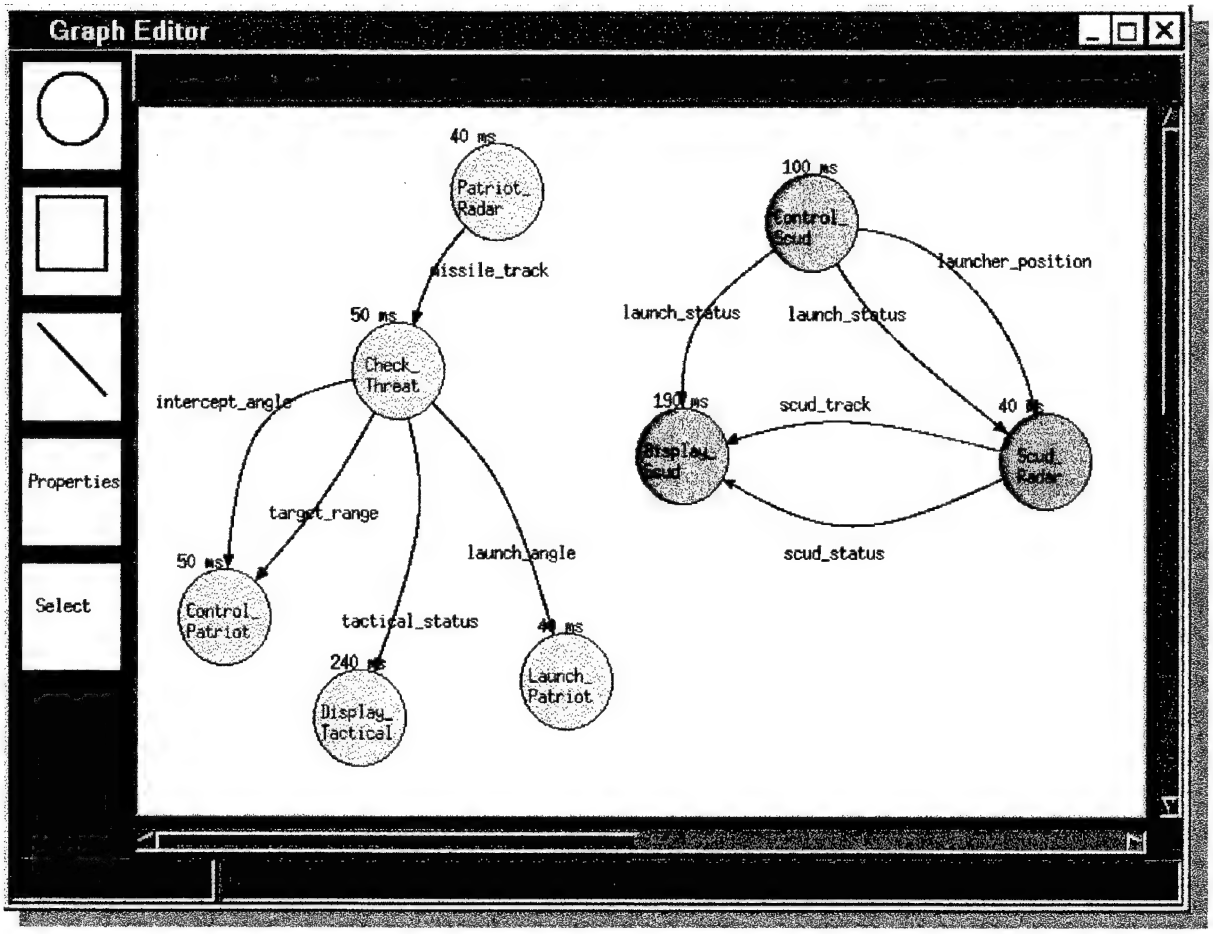



**Figure 2-1. CAPS Graphical User Interface.**

As a result, code that incorporates global communication and real-time control is automatically generated. This capability overcomes the uncertainty and brittleness that is intrinsic in manually scheduled, real-time software.

In addition, CAPS will provide automated assistance for finding reusable code. This is accomplished via

- keyword searches (currently implemented),
- subtype matches (planned),
- semantic matches (planned),

performed at the level of the software specification and at the level of the assets under configuration management.

In summary, CAPS provides a powerful prototyping and software development tool that allows users to quickly address software prototyping, development, and/or modeling needs. It is especially suited to addressing software development and/or modeling that involves concurrent processes.

## 2.2 CAPS Operation

CAPS is currently developed to run on SUN/Unix workstations. Figure 2-2 shows a high-level view of the CAPS process. It illustrates the fact that CAPS is a human-in-the-loop process that expedites the development of software but still requires analyst input and control. The more detailed process chart shown in Figure 2-3 depicts the various steps one follows in applying CAPS to generate prototype software and/or system models. The diagram is generic in the sense that it applies to both the development of entirely new system representation and the modification of existing system representations. In the diagram, the rectangles (which are identified with numbers) represent actual CAPS steps. The rounded rectangles (which are identified with letters) represent steps outside of CAPS needed to implement the CAPS process.

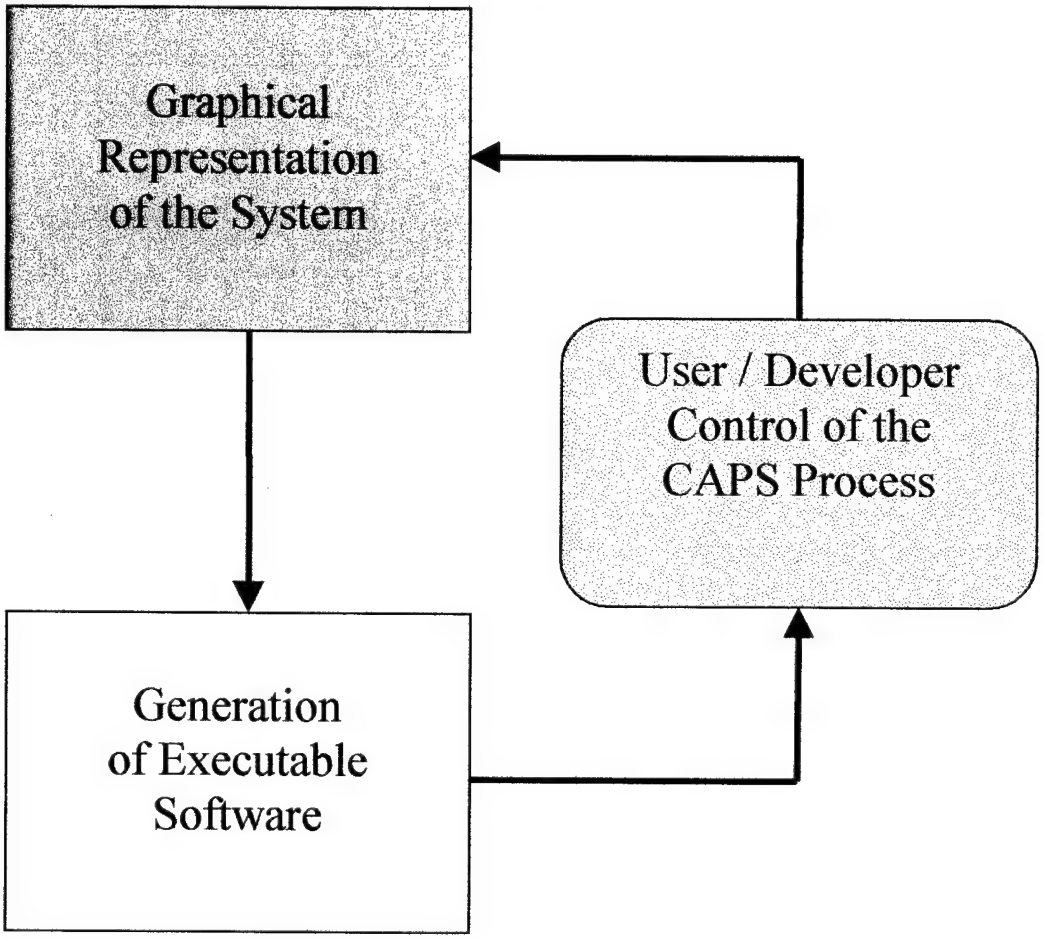



**Figure 2-2. Overview of CAPS Process.**

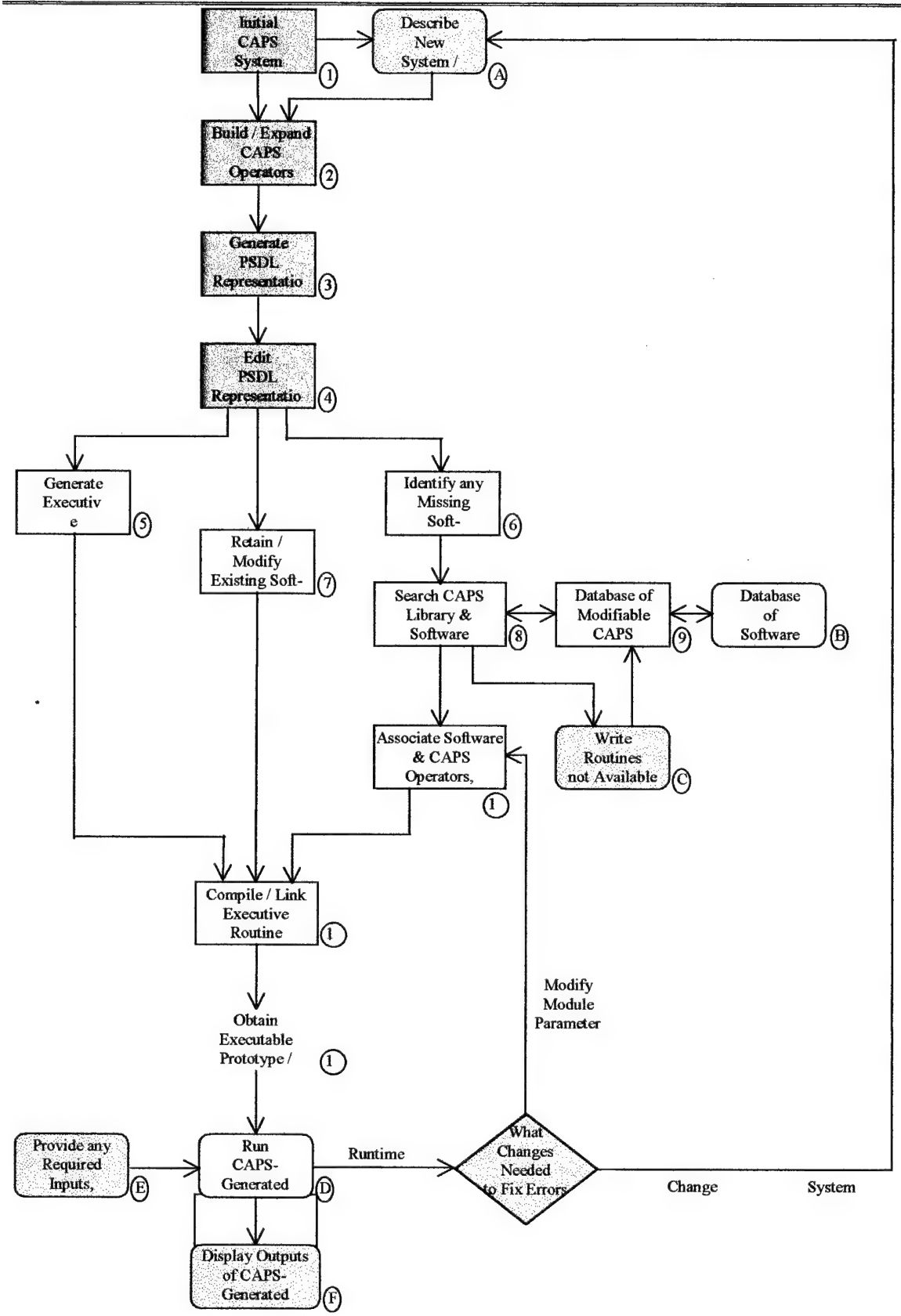



**Figure 2-3. Details of the CAPS Process.**

Assuming a CAPS system description already exists, then the top rectangle in Figure 2-3 (box 1) will involve initially calling up that system description within CAPS. If one does not exist, then the CAPS process will begin with a blank screen. In either case, the user must describe the new system, or the modifications to the existing system, that will be represented by the soon-to-be-created or modified CAPS software (box A). This new or modified description will be used to guide the creation or modification of the CAPS input screen "Operators and Streams" diagram (box 2). The "Operators and Streams" diagram identifies the software modules used to represent the system and the interconnection and timing relationships between/among the nodes. CAPS then uses the "Operators and Streams" diagram to produce a PSDL representation of the system (box 3). CAPS also provides a text editor to allow the CAPS user to edit the PSDL representation, as required (box 4). CAPS then processes the edited PSDL representation to generate the executive software for the system (box 5), identify any missing software modules (box 6), and retain software modules from the original system representation, as appropriate (box 7).

The retained software modules may also have to be modified to correspond to the specific system functions/parameters being represented. For the missing modules, CAPS will search the CAPS library of modifiable software routines (boxes 8 and 9). If the missing module is not available within the CAPS library, CAPS will also examine a database of software utilities (box B) that could be accessed by the CAPS executable software during the execution of the CAPS-generated system software. The CAPS library will contain software wrappers for accessing the utility software by the CAPS-generated executable software. If library or utility software for any missing modules is not available, then the CAPS user will have to develop new software routines (box C), which will then be inserted into the CAPS database. Once the missing software modules are available, CAPS will associate the library routines with the CAPS "Operators and Streams" nodes (box 10). At this point, the CAPS user may also have to make minor modifications to the generic library software to ensure that it represents the desired system characteristics (e.g., setting the system gain). The executive software, the retained software modules and the new software modules will be complied and linked together (box 11) and the executable software will be generated (box 12). The resulting executable software will be run (box D) with any user-provided inputs or controls (box E). The CAPS executive will ensure that the CAPS executable software reports any required runtime errors, should there be any problems with the CAPS-defined system or module parameters. The CAPS user should then make appropriate modifications to the system description and/or module parameters. The CAPS-generated software's outputs can also be displayed graphically (box F).

## 2.3 CAPS Assessment/Limitations

CAPS represents a potentially valuable tool for expediting the development of prototype system software. However, in addition to this originally envisioned role, CAPS can also support the development of models and simulations of systems. In this latter role CAPS would be used to generate software that represents the entire operation of a system (or even system of systems) as opposed to merely the embedded software within a system.

There were many participants in the CAPS development process. The major participants in the STTR (Phases I and II) were:

- Illgen Simulation Technologies, Inc. (ISTI) – the prime contractor,
- Monmouth University (MU) – the major subcontractor,
- Drs. Luqi and Berzen of the Naval Post Graduate School (NPGS) – consultants.

One of the significant accomplishments over the past six months consisted of integrating all of the various pieces of CAPS and getting it to run properly. In addition, a few critical features, originally envisioned for CAPS, were only developed during the latter months of the CAPS contractual effort. For example, CAPS was always envisioned to have access to a library of software routines that a user could plug into his CAPS representation of a system. ISTI developed software to enable a CAPS user to retrieve previously developed software modules and integrate them into the CAPS-produced software. ISTI designed this capability to support the importation of foreign (i.e., non-CAPS-specific) software from other sources. For example, a CAPS user can now import routines from commercial software products such as MATLAB or from legacy software. The only requirement is that the software would have to support the CAPS software interface, which might necessitate writing a "wrapper" around the modules. Also, the version of CAPS originally delivered to ISTI had a proprietary PSDL editor. Moreover, this PSDL editor produced recompilations of the PSDL code every time a single line was changed, which slowed down the CAPS process. To address both of these issues, ISTI developed its own PSDL editor.

While CAPS represents a potentially powerful software development tool, the current version has a few limitations as discussed below:

- Generation of Ada code – CAPS generates Ada code that is not widely used by the software industry. The decision to use Ada was made by the NPGS during the original formulation of the CAPS concept. Ada provides the advantage that it enforces a level of internal consistency within the software.
- GUI operation – CAPS provides a GUI that allows a CAPS user to visualize the interrelationships among the system modules. Unfortunately, the original GUI development occurred many years ago so the CAPS GUIs do not follow current industry standard (e.g., Microsoft, X-Windows) GUI practices. Also, information that is entered graphically doesn't always result in an automatic update of the textual PSDL code. This is especially true when modifying (i.e., editing) a flow diagram.
- Speed of operation – CAPS sometimes seems to run slowly as a result of its need to redraw the entire diagram when only a small portion is changed and to recompile the PSDL code every time a line entry change is made.
- Generation of executive code – CAPS concentrates on the generation of executive software needed to control the concurrent processes involved with the prototype software. This represents one of, and often *the*, most critical element of the code in a real-time system. Unfortunately, CAPS does not also automatically generate code for those operations performed within the lowest-level elements in the system diagram.
- Consistency checking – CAPS is particularly attentive to testing for timing inconsistencies within the process diagram, the PSDL code, and the ultimate output software.

The above factors will be discussed further in the section of this document that describes possible future efforts with CAPS.

One of the lingering concerns about CAPS concerns its status versus potential competitors. Because of CAPS long development history, other software now exists that performs many of the features that were originally major innovations when CAPS was originally conceived. This will also be discussed in the section describing possible future efforts with CAPS.

# 3.0 APPLICATION OF CAPS TO A SAMPLE PROBLEM

This section describes the background associated with and the results of applying CAPS to a sample problem.

## 3.1 Rationale for Selection of the Sample Problem

The original version of CAPS provided by the NPGS included a few simple example applications such as the Patriot missile flyout. ISTI determined that to more fully test and demonstrate CAPS capabilities, it needed to develop its own sample problem(s). Because the final CAPS development and integration occurred in the ISTI office in Arlington, VA, the nearby Night Vision Electronic Sensors Directorate (NVESD) at Ft. Belvoir was chosen as a facility with potentially challenging and appropriate prototyping and modeling and simulation requirements. Moreover, ISTI had recently received a contract from the NVESD (through BDM) for high-tech engineering support services.

After meeting with several groups within the NVESD and presenting CAPS capabilities, ISTI identified a sample application involving analysis of the hardware design needed to implement an automatic target recognition (ATR) algorithm. The problem appeared amenable to simulation and would therefore represent a fair test of CAPS' ability to facilitate the development of system simulations.

## 3.2 Description of the Sample Problem

ISTI was given a report that described the ATR algorithm and was asked to determine how many Power PC processors would be required to implement that algorithm within the available time for the assumed focal plane array (FPA). ISTI was told to assume that the FPA generated a 1315 by 480-pixel array, with each pixel encoded on a 12-bit (4096-level) gray scale. Further, the FPA was assumed to be able to generate the IR scene at a 30-Hz rate. However, the NVESD asked us to base our analysis on the need to process the algorithm at a 10-Hz rate. In addition to calculating the required number of processors, ISTI was asked to generate bus and processor utilization statistics.

The ATR algorithm involved 7 levels of computation (called rounds 0 through 6). The round 0 portion of the algorithm consisted of 6 target masks that had to be applied to *each* pixel in the array. The target masks consisted of calculating an average IR intensity for 30 surrounding pixels near the candidate pixel and 30 additional surrounding pixels that would reside outside the target's boundaries. A target would be declared to exist if a sufficient contrast existed between the two averages. (If a pixel was not centered on an actual target, then the chances of this ratio exceeding a pre-set threshold would be low. If a pixel was centered on a candidate target, then the ratio would more likely be exceeded.) The purpose of the round 0 processing was to determine candidate target pixel locations for each of the 6 assumed target types. One can see that the round zero process involved calculating 2 30-pixel averages and performing a threshold comparison for each of the 631,200 pixels in the FPA within 0.1 seconds – an impressive number of computations. Once the round 0 candidates were identified, they were then subjected to further round 1 processing, which involved the application of 7 target masks and somewhat more

detailed calculations. This culling process continues until all 7 levels of the ATR algorithm are processed. Each round involves increasingly more detailed computations, but on a decreasing number of candidates. As a result, over 90% of the total computational workload is associated with round 0 and about 99% for rounds 0 and 1. With the concurrence of the NVESD, ISTI concentrated on simulating only the first two rounds of the ATR algorithm.

Based on our review of the ATR algorithm, ISTI developed an implementation strategy that emphasized computational efficiencies for round zero. These included

- Modification of the micro code to accelerate computations,
- Adjustments to the algorithm to avoid floating point divisions,
- Maximum use of processor cache memory to speed the movement of data,
- Sharing of computations equally among all processors, to the maximum extent possible.

ISTI also used these same techniques for round 1 processing to the maximum extent possible.

ISTI's CAPS simulation assumed separate processors[1] dedicated to rounds 0 and 1. We also assumed that the round 0 processors would send candidate pixel locations to the round 1 processors as the candidates were found. This approach involves the following:

- Permits simultaneous processing of rounds 0 and 1,
- Requires the establishment of thousands of cross bar reconfigurations (one each time a round 0 candidate is detected),
- Creates potential bus conflicts and idling of the round 0 processors,
- Results in the potential for unbalanced round 1 workload,
- Produces unbalanced round 0 processing time when an unbalanced distribution of round 0 candidates occurs within the FPA scene.

The CAPS-generated simulation addressed these issues.

## 3.3 Results of the Sample Problem Analysis

The CAPS evaluation had to pay special attention to the sharing on data between the round 0 and round 1 processors because this created the potential for inefficiencies in the ATR algorithm implementation. Figure 3-1 shows the crossbar switching cases that were included in the simulation. In this figure, each circle represents a separate processor. Each of the 4 panels in the figure assumes a total of 32 processors. The processors are interconnected in a hierarchical manner wherein 4 adjacent processors need only go up one level in the switching network to share data. The processors within the local group of 16 processors need go up two levels to share data, while three levels are required to share data between processors in different local groups. This illustration assumes that the right most processor in each local group of 16 is

[1] To complete our analysis of the problem, ISTI also examined an approach wherein all of the processors were initially dedicated to round 0 and at the conclusion of these calculations, shared results and began processing round 1. This alternate approach was not simulated.

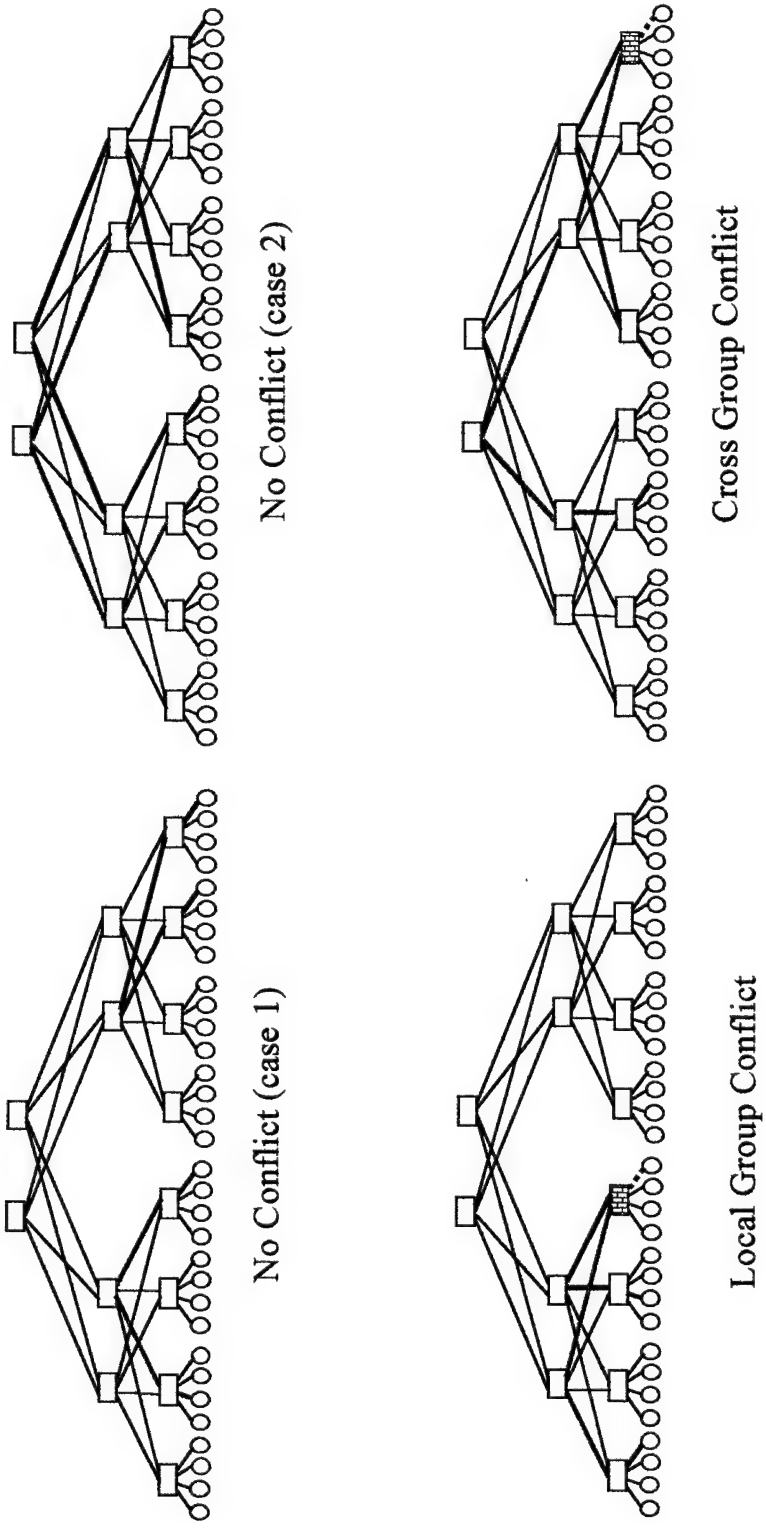



Figure 3-1 Cross-Bar Switching Cases

**Figure 3-1. CrossBar Switching Cases.**

dedicated to round 1 processing and the other 15 processors perform round 0 processing. Because only two round 1 processors are assumed, there is a 50% chance of a conflict arising when two of the round 0 processors simultaneously have identified a candidate pixel.

The upper left panel shows a no-conflict case where two processors in one local group and two other processors in the other local group need to share data simultaneously. The upper right panel shows the case where two processors in one local group need to share simultaneously data with processors in the opposite local group. Conversely, the upper right panel could represent two processors in separate local groups needing to share data simultaneously with processors in the opposite local group. In either of these cases, a crossbar bus conflict does not occur. In the lower left panel, two processors in the same local group need to share data with the same processor. In the lower right panel two processors in different local groups need to share data with the same processor. In both of these latter cases, a crossbar conflict does occur, as shown by the shared switch in these two panels. Obviously, when additional simultaneous communications need to occur, the chance of crossbar conflicts is increased. Because the operation of the crossbar switch was of great interest to the NVESD (because it can be a limiting factor in the system design), the CAPS simulation had to represent these problems. Moreover, establishing a crossbar switching path takes considerable time (on the order of 125 nanoseconds for each level of switching[2] as opposed to 5 nanoseconds to transfer a data word) and can greatly slow down the processing.

The CAPS design assumed that each of the round 0 processors was assigned an equal part of the FPA scene to process. For the 30 round 0 processors shown in Figure 3-1, each would have approximately 21,000 pixels to examine as potential target locations. However, because the actual candidates detected during the round 0 processing could occur anywhere, the 30 round 0 processors essentially randomly find candidates to pass to the round 1 processors. Therefore, the CAPS simulation addressed the time required to perform the round 0 processing, the random occurrence of round 0 candidates and the subsequent contention for the crossbar switching, and the time for round 1 processing.

## 3.4 Improvements Necessitated by the CAPS Sample Problem

Two important CAPS limitations surfaced while working the sample problem. The first involved the time scale available within CAPS. CAPS originally only supported millisecond time scale granularity. However, the sample problem involved nanosecond granularity. We had two options. The first was to merely re-map the CAPS time scale (i.e., interpret 1 millisecond as representing 1nanaosecond). The other option was to modify CAPS to support the necessary time scaling. The second problem encountered involved simplistic interdependencies within CAPS. The version of CAPS provided to ISTI did introduce time delays for performing specific functions. However, it did not support situations in which the time to complete one function could vary depending upon the completion of some other function. That is, the time

[2] This means that the crossbar switch establishment time for transferring data between processors in different local groups is 5 x 125, or 625 nanoseconds. This time can add up quickly if a larger number of switch connections are needed. Remember that the entire ATR algorithm has to be processed for the entire FPA within 0.1 seconds, or 100,000 microseconds.

dependencies were static and not dynamic. For the sample problem chosen, it was imperative that the latter situation[3] be addressed. As a result of identifying these two problems, ISTI significantly modified CAPS so it could address the needs of the sample problem. However, due to the late date when these problems were discovered, the revised version of CAPS concentrated on producing the necessary results but did not incorporate the GUI features associated with the baseline version of CAPS.

## 3.5 Observations from the Sample Problem Evaluation

The examples provided with the original version of CAPS were largely contrived to take advantage of the features CAPS possessed. However, the sample problem was not designed nor especially chosen to mimic CAPS capabilities. Rather, it represented a real-world problem that helped determine specific limitations in the original version of CAPS. The sample problem was relatively complex, which meant that it stressed CAPS' capabilities. However, this complexity meant that it required all of the time and resources available for beta testing of CAPS. Ideally, CAPS should be subjected to additional testing to better determine its true capabilities and limitations.

The CAPS-generated simulation of the ATR implementation provided statistics on the operation of the proposed multi-processor design. Figure 3-2 presents an illustration of the potential crossbar switching conflicts for the 32-processor configuration shown in Figure 3-1. The histogram in Figure 3-2 shows the times when 0,1, or 2 round 0 processors have simultaneous candidate target locations to pass to the 2 round 1 processors. The figure actually depicts the results for a portion of the 0.1 second time period available for round 0 processing. Because of the assumed design, cases of three or more bus conflicts do not occur; rather, the additional processors merely wait in a queue until they can access the crossbar switch. Because the design assumed that a round 0 processor would have to suspend further round 0 processing until a candidate it identifies is passed to a round 1 processor, the processor utilizations were far below an optimal 100%, as shown in Table 3-1.

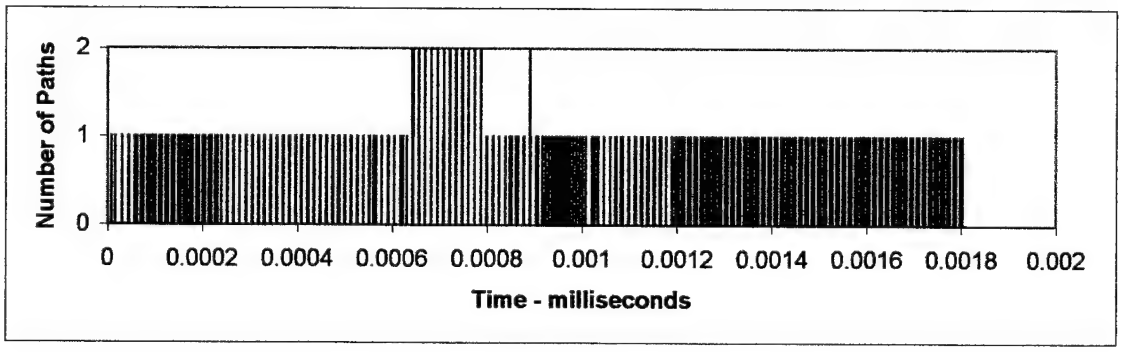



**Figure 3-2. Crossbar Switch Utilization.**

[3] For the ATR implementation analysis, it was necessary to analyze the impact of a processor having to wait on the availability of the crossbar-switching network. This meant that round 0 processing times could vary depending upon what else was going on within the overall system.

**Table 3-1. Processor Idle Times.**

| | Round 0 Processors | Round 1 Processors |
|---|---|---|
| Mean | 28.85473% | 9.043% |
| Standard Development | 0.000123% | 3.2156% |
| Minimum | 28.85449% | 6.769% |
| Maximum | 28.85501% | 11.317% |

## 3.6 Observations from the CAPS Sample Problem

Two important CAPS limitations surfaced while working the sample problem. The first involved the time scale available within CAPS. CAPS originally only supported millisecond time scale granularity. However, the sample problem involved nanosecond granularity. We had two options. The first was to merely re-map the CAPS time scale (i.e., interpret 1 millisecond as representing 1nanaosecond). The other option was to modify CAPS to support the necessary time scaling. The second problem encountered involved simplistic interdependencies within CAPS. The version of CAPS provided to ISTI did introduce time delays for performing specific functions. However, it did not support situations in which the time to complete one function could vary depending upon the completion of some other function. For the sample problem chosen, it was imperative that the latter situation[4] be addressed. As a result of identifying these two problems, ISTI significantly modified CAPS so it could address the needs of the sample problem. However, due to the late date when these problems were discovered, the revised version of CAPS concentrated on producing the necessary results but did not incorporate the GUI features associated with the baseline version of CAPS.

[4] For the ATR implementation analysis, it was necessary to analyze the impact of a processor having to wait on the availability of the crossbar-switching network. This meant that round 0 processing times could vary depending upon what else was going on within the overall system.

# 4.0 CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE DEVELOPMENT AND IMPROVEMENT OF CAPS

## 4.1 Conclusions

While CAPS is now able to generate executable code and while it possesses significant features not found in the original version, CAPS appears to need some further development and application before it could be offered as a stand-alone software product. This section discusses the options and ideas ISTI has formulated to finalize the commercialization of CAPS.

## 4.2 Recommendations

CAPS chances of commercial success would be greatly enhanced if the following further developments were accomplished:

1. Improved GUIs – The software industry has standardized on Microsoft Windows and Unix-based XWINDOWS GUI features. CAPS began before these GUI features were as well refined as they are today. Therefore, the original CAPS developers created their own GUI process. Unfortunately the original CAPS GUI developers did not fully anticipate where Windows and XWINDOWS were headed. Because of the need to address more pressing issues concerning CAPS, ISTI has not as yet been able to revise and update the CAPS GUIs. In addition to being non-standard, the CAPS GUIs are awkward to use. For example, the entire graphical screen is re-generated every time a small change is made to a CAPS process diagram. Also, CAPS lacks an automatic linkage between the PSDL text editor and the process diagrams.

2. Generation of C code – CAPS produces Ada code. When CAPS development began, Ada was much more prominent, at least within the DoD. Moreover, Ada represented a good choice because it provides internal error checking, which can be especially important when trying to develop software rapidly that also accurately represents the intended processes. Unfortunately for CAPS, the C programming language is much more widely used throughout the software industry. Therefore, the commercial viability of CAPS would be enhanced if it could generate C code. Except for some preliminary efforts to incorporate C code in the sample problems analysis, ISTI did not have the time to totally rewrite the CAPS system to generate C code.

3. Improved interdependencies –While analyzing the sample problem, it became apparent that there are some real-world dynamic dependencies within systems that can't readily be addressed by CAPS. For example, the speed of operation of (or the ability to perform) one process could be affected by the results of some earlier process. To proceed with the sample problem analysis, ISTI developed a work-around for this limitation. However, this workaround has not been totally integrated into the CAPS operating system, but should be if CAPS is to be made more commercially useful.

4. Conversion to run on PCs – With the further rapid advances in computer technology, today's PCs provide levels of performance that until just recently could only be obtained

with higher-priced workstations. Unfortunately, CAPS is designed to run on SUN workstations. The commercial utility of CAPS would be greatly enhanced if, as part of making the other revisions cited above, the CAPS operating software were modified so it could run on PCs.

While CAPS began as a break-through concept for expediting the development of system prototypes (and system models and simulations), the software industry has caught up with, and in many areas surpassed, CAPS since its original formulation. Therefore, ISTI views the improvements above cited improvements as essential to full commercialization of CAPS.

Because ISTI has already invested a considerable amount of its own resources into CAPS, additional company-funded development with little immediate direct application appears unwise. Therefore, ISTI has developed a strategy wherein we could begin applying CAPS on a more limited basis and in a more tightly controlled environment than if it were offered as shrink-wrapped software. In this alternative approach, ISTI would use CAPS as an in-house software development tool. This strategy has the following three elements:

1. Use CAPS to support other ISTI projects – ISTI has a number of projects that involve some level of software development. CAPS could be employed by ISTI personnel to develop the software models and simulations required by these other corporate projects. This would ensure that the CAPS users would have, or have immediate access to, the ISTI personnel who have most recently been involved with the development and application of CAPS. These personnel would be able to work around the current limitations within CAPS and also make further upgrades to CAPS in areas such as those cited above during the natural course of their use of CAPS.

2. Use CAPS to support SimCentral – ISTI is developing a commercial Web-based modeling and simulation service known as SimCentral. SimCentral will allow the public to access information about modeling and simulation. SimCentral will also allow subscribers to have access to additional databases and to a wide range of modeling and simulation experts. CAPS could become a natural extension of the SimCentral product line wherein SimCentral subscribers could ask to have specific software routines developed by the SimCentral support staff. The SimCentral support staff could then employ CAPS to develop the software modules being requested by the SimCentral subscribers.

3. Populate the CAPS software module library – CAPS would become a much more useful software development tool if it provided an extensive library of software routines. CAPS users could then insert these library routines into their over system models and simulations. By pursuing the above two opportunities to utilize CAPS within ISTI, we would have a natural opportunity to populate the CAPS library. The library would then contain both CAPS-specific software and CAPS interface software to provide access to other sources of software modules (e.g., MATLAB).

Once the above activities are accomplished, CAPS would have an improved library, improved GUIs, and extensive testing/trial application. The only remaining developmental item that would

need attention would involve the preparation of suitable documentation. Ideally this would include introducing into CAPS extensive help menus and files. At that point, CAPS would be ready for release as an off-the-shelf software tool that ISTI could offer to others.